

Table of Contents

1. Introduction.....	1
1.1. Problem Statement.....	1
1.2. Purpose & Goals	1
1.3. System Overview.....	2
2. Design & Architecture.....	3
2.1. User Interface.....	5
2.2. Features.....	5
2.3. Tools and Techniques Used.....	6
2.4. Implementation.....	7
2.5. Final Result.....	12
3. Future Works.....	13
4. Conclusion.....	13

1. Introduction

Managing personal finances can be challenging without the right tools. The **Finance Tracker** project is designed to help users track their income and expenses, analyze spending habits, and make better financial decisions. Using Python, Flask, and libraries like Pandas and Matplotlib, the tracker offers a user-friendly interface with features such as categorized transactions, visual reports, and data summaries. This project aims to simplify financial management and empower users to achieve their financial goals.

1.1. Problem Statement

In today's fast-paced world, many individuals struggle to manage their finances effectively due to the lack of an efficient and accessible tool for tracking income and expenses. Traditional methods like manual record-keeping or relying solely on bank statements can be time-consuming, error-prone, and insufficient for analyzing spending patterns.

This lack of proper financial management often leads to overspending, poor budgeting, and missed financial goals. There is a need for a simple, intuitive, and automated system that allows users to track their financial activities, categorize transactions, and gain insights into their spending habits to make informed decisions.

The **Finance Tracker** addresses this problem by providing an easy-to-use platform for managing personal finances, helping users stay organized and achieve better financial control.

1.2. Purpose & Goals

Purpose:

The primary purpose of the **Finance Tracker** project is to provide users with an efficient and user-friendly tool to manage their personal finances. By offering features to record, categorize, and analyze financial transactions, the application aims to simplify the process of tracking income and expenses, enabling users to make informed financial decisions and achieve their budgeting goals.

Goals:

- 1) Expense and Income Tracking
- 2) Simple and User-Friendly Interface
- 3) Improved Financial Awareness
- 4) Scalability and Future Enhancements

1.3. System Overview

The **Finance Tracker** is a web-based application designed to help users effectively manage their personal finances. The system allows users to record their income and expenses, categorize transactions, and analyze financial data through visual summaries such as graphs and charts.

Key Components

1. User Interface:

- a. A user-friendly front-end developed with Flask and HTML/CSS, enabling users to input data and view reports easily.

2. Database:

- a. A structured database for securely storing transaction details, categorized as income or expenses, along with timestamps and additional notes.

3. Data Processing:

- a. Python libraries such as Pandas are used to process and organize financial data for analysis.

4. Data Visualization:

- a. Tools like Matplotlib are utilized to generate charts and graphs, helping users understand their financial habits visually.

5. System Workflow:

- a. Users input their financial transactions through the interface.
- b. The data is stored in the database and processed for categorization and summary generation.
- c. Visualizations and summaries are presented to the users, providing actionable insights.

The system is designed to be scalable, with the potential for integrating advanced features like budget forecasting, automated reminders, and API connections to banking systems in future iterations. This makes the **Finance Tracker** a reliable tool for financial management and decision-making.

2. Design And Architecture

The **Finance Tracker** application is built with a modular design and follows a three-tier architecture, ensuring scalability, maintainability, and efficiency. The system's components are structured to handle user interactions, data processing, and storage effectively.

1. Architectural Overview

- **Presentation Layer (Front-End):**
 - Developed using Flask with HTML, CSS, and JavaScript.
 - Provides an intuitive interface for users to input transactions, view summaries, and analyze financial data.
- **Application Layer (Back-End):**
 - Built with Python and Flask.
 - Manages business logic, processes data, and handles requests between the front-end and the database.
- **Data Layer (Database):**
 - Uses a relational database (e.g., SQLite or SQL Server).
 - Stores transaction details, categories, and user data in a secure and structured format.

2. System Design

- **ER Diagram:**
 - Entities:
 - User: Stores user credentials and preferences.
 - Transaction: Captures income/expense details, category, amount, date, and description.
 - Category: Predefined or user-defined categories for transactions.
 - Relationships:
 - A user can have multiple transactions.
 - Each transaction belongs to a category.

- **Database Schema:**

- **User Table:**

- User_ID (Primary Key)
 - Name
 - Email
 - Password

- **Transaction Table:**

- Transaction_ID (Primary Key)
 - User_ID (Foreign Key)
 - Category_ID (Foreign Key)
 - Amount
 - Date
 - Description

- **Category Table:**

- Category_ID (Primary Key)
 - Name

3. Workflow

1. **Data Input:** Users add income or expense entries through the web interface.
2. **Data Processing:** Input data is validated and stored in the database. Python processes data for reports and summaries.
3. **Data Visualization:** Charts and graphs (e.g., bar charts for expenses by category) are generated using Matplotlib.
4. **Output:** Summarized financial insights are displayed to the user in an easy-to-understand format.

4. Design Diagrams

- **Data Flow Diagram (DFD):**

- Level 0: User interacts with the system.
 - Level 1: Transactions flow from input to database storage and visualization generation.
 - Level 2: Processed data is used to generate charts and insights.

- **System Architecture Diagram:**

- Showcases interaction between the front-end, back-end, and database.

This design ensures that the **Finance Tracker** is robust, user-centric, and adaptable for future enhancements. Would you like help with diagrams or a more detailed explanation?

2.1. User Interface

The **Finance Tracker** offers a user-friendly and visually appealing interface designed to simplify financial management. The homepage serves as a centralized dashboard where users can view an overview of their financial activity, including total income, expenses, recent transactions, and visual insights such as pie charts and bar graphs. This provides users with a clear understanding of their financial status at a glance.

The application includes a dedicated page for adding transactions, featuring an intuitive form where users can input transaction details such as amount, date, description, and category. They can also specify whether the transaction is an income or an expense. The transaction history page displays a comprehensive list of all past entries in a clean and organized manner. Users can filter transactions by date, category, or type and have the option to edit or delete individual records directly from this page.

Designed with simplicity and responsiveness in mind, the interface adapts to different devices, ensuring an optimal user experience on desktops, tablets, and smartphones. Accessibility features such as clear labels, tooltips, and feedback messages for user actions enhance the overall usability. The Finance Tracker interface ensures that users can easily interact with the system, stay organized, and gain valuable insights into their financial activities.

2.2. Features

The **Finance Tracker** is equipped with a range of features designed to simplify personal financial management. These features ensure that users can effectively monitor their income and expenses, gain insights into their financial habits, and make informed decisions.

The application allows users to record and categorize transactions seamlessly. Whether it's income from a salary or expenses like groceries and rent, users can input these details with associated categories, dates, and optional descriptions. This ensures that all financial data is organized and easy to retrieve. Visual reports and summaries are integral to the Finance Tracker.

Users can access charts and graphs that illustrate their spending patterns, income trends, and category-wise expense breakdowns. These visualizations provide a quick and intuitive way to analyze financial data and identify areas for improvement. A transaction history section offers a complete record of all past transactions.

2.3. Tools & Techniques Used

1. **Flask:**

A lightweight Python web framework used to build the core of the application. It handles routing, user requests, and template rendering, providing flexibility and scalability.

2. **HTML/CSS:**

Used for structuring and styling the user interface. HTML defines the structure of the web pages, while CSS ensures the layout is clean, professional, and visually appealing.

3. **Bootstrap:**

A responsive design framework incorporated to ensure the application adapts to various screen sizes, making it accessible on desktops, tablets, and smartphones.

4. **Pandas:**

A powerful Python library used for data manipulation and analysis. It helps in organizing transaction data, performing calculations, and generating summaries for financial insights.

5. **Matplotlib:**

A Python library used to create static, interactive, and animated visualizations. It is used for generating graphs and charts to visualize financial data, such as spending trends and income/expense comparisons.

6. **Chart.js:**

An alternative to Matplotlib for creating interactive and aesthetically pleasing charts. It enhances user engagement and provides a more dynamic view of financial data.

7. **SQLite/SQL Server:**

A relational database used to store transaction data. It ensures structured storage of information, making it easy to retrieve and manipulate for reports and analysis.

8. **JavaScript:**

Used for adding interactivity to the web interface, such as form validation, dynamic data input handling, and interactive chart features.

2.4. Implementation

The **Finance Tracker** is implemented using a combination of front-end and back-end technologies to provide a seamless experience for users to manage and visualize their financial data.

1. Front-End Implementation

The front-end of the application is developed using **HTML** and **CSS** to structure and style the user interface. HTML forms are used to capture user inputs for financial transactions, while CSS ensures that the interface is clean, responsive, and user-friendly. **Bootstrap** is integrated into the design to make the application mobile-responsive, ensuring it functions well across different devices.

```

1  <doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1">
6    <title>Finance Tracer</title>
7    <link rel="icon" type="image/png" href="/static/icon.png">
8    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
9    <style>
10
11  </style>
12 </head>
13 <body>
14
15
16   <nav class="navbar navbar-expand-lg bg-black">
17     <div class="container-fluid">
18       <a class="navbar-brand text-white" href="/">Finance Tracer</a>
19       <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
20         <span class="navbar-toggler-icon"></span>
21       </button>
22       <div class="collapse navbar-collapse" id="navbarSupportedContent">
23         <ul class="navbar-nav me-auto mb-2 mb-lg-0">
24           <li class="nav-item">
25             <a class="nav-link text-white" href="/about">About</a>
26           </li>
27           <li>
28             <a class="nav-link text-white" href="/register">Register</a>
29           </li>
30         </ul>
31       </div>
32     </div>
33   </nav>
34
35   {% block content %}
36
37   {% endblock content %}
38
39
40   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-YvpcrVf0tY3lH66BddM986W41+zN9oum9Qv6327ybAsm9ZvN43Xdpfp6" crossorigin="anonymous"></script>
41 </body>
42 </html>

```

```
home.html
templates > home.html > section#bg_min-vh-100 > div.container > div.row.justify-content-center > div.col-lg-3 > form > button.btn.btn-primary
1 {% extends 'base.html' %}
2 {% block content %}
3
4 <section id="bg" class="min-vh-100">
5   <div class="container">
6     <div class="row justify-content-center">
7       <div class="col-lg-3">
8         <form action="/" method="POST">
9           <div class="mb-3">
10             <label class="form-label">Account Name</label>
11             <input type="text" class="form-control" name="AccountName">
12             <div id="emailHelp" class="form-text">Please register first if you don't have an account.</div>
13           </div>
14           <div class="mb-3">
15             <label for="exampleInputPassword1" class="form-label">Password</label>
16             <input type="password" class="form-control" id="exampleInputPassword1" name="Password">
17           </div>
18           <button type="submit" class="btn btn-primary">Login</button>
19         </form>
20       </div>
21     </div>
22   </div>
23 </section>
24
25
26 {% endblock content %}
27
28
```



```

register.html X
templates > register.html > div > div.row.justify-content-center.mt-2 > div.col-lg-3 > form
1  {% extends 'base.html' %}
2  {% block content %}
3
4  <div>
5      <div class="row justify-content-center mt-2">
6          <div class="col-lg-3">
7              <form action="/Register" method="POST">
8                  <div class="mb-3">
9                      <label class="form-label">Account Name</label>
10                     <input type="text" class="form-control" name="AccountName">
11                 </div>
12                 <div class="mb-3">
13                     <label for="exampleInputPassword1" class="form-label">Password</label>
14                     <input type="password" class="form-control" id="exampleInputPassword1" name="Password">
15                 </div>
16                 <button type="submit" class="btn btn-primary">Register</button>
17             </form>
18         </div>
19     </div>
20 </div>
21
22 {% endblock content %}
23
24

```

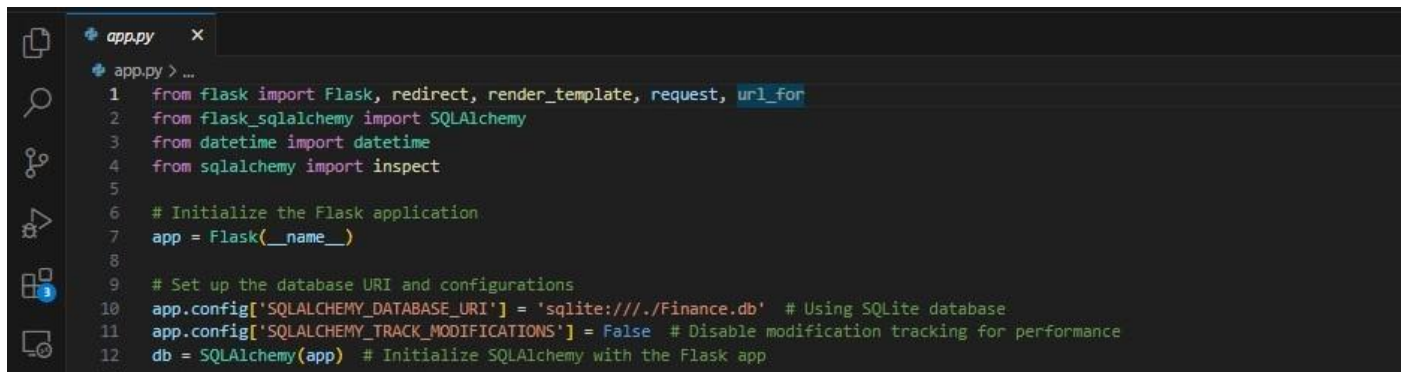
```

account.html X
templates > account.html > div.mb-5 > table.table.table-dark.table-striped > tbody
4  <div class="row justify-content-center mt-2">
5      <div class="col-lg-3">
6          <form action="/[[ AccountName ]]" method="POST">
7              <div class="input-group mb-3">
8                  <span class="input-group-text">$/span>
9                  <input type="number" class="form-control" name="Withdraw" placeholder="Enter the amount..." aria-label="Amount (to the nearest dollar)">
10                 <span class="input-group-text">.<span>00</span>/span>
11             </div>
12             <div class="mb-3">
13                 <label class="form-label">Description</label>
14                 <input type="text" class="form-control" name="Description">
15             </div>
16             <button type="submit" class="btn btn-primary">Withdraw</button>
17         </form>
18     </div>
19
20     <div class="col-lg-3">
21         <form action="/[[ AccountName ]]" method="POST">
22             <label class="form-label">Amount</label>
23             <div class="input-group mb-3">
24                 <span class="input-group-text">$/span>
25                 <input type="number" class="form-control" name="Deposit" placeholder="Enter the amount..." aria-label="Amount (to the nearest dollar)">
26                 <span class="input-group-text">.<span>00</span>/span>
27             </div>
28             <div class="mb-3">
29                 <label class="form-label">Description</label>
30                 <input type="text" class="form-control" name="Description">
31             </div>
32             <button type="submit" class="btn btn-primary">Deposit</button>
33         </form>
34     </div>
35 </div>
36
37 <div class="mt-5">
38     <table class="table table-dark table-striped">
39         <thead>
40             <tr>
41                 <th scope="col">S.No.</th>
42                 <th scope="col">Date</th>
43                 <th scope="col">Description</th>
44                 <th scope="col">Action</th>
45                 <th scope="col">Amount</th>
46                 <th scope="col">Balance</th>
47             </tr>
48         </thead>
49         <tbody>
50             <tr>
51                 <th scope="row">{{loop.index}}</th>
52                 <td>{{records.date}}</td>
53                 <td>{{records.desc}}</td>
54                 <td>{{records.op}}</td>
55                 <td>{{records.amount}}</td>
56                 <td>{{records.balance}}</td>
57             </tr>
58             <tr>
59                 <th colspan="6">{{for records in transactions}}
60                     <tr>
61                         <th scope="row">{{loop.index}}</th>
62                         <td>{{records.date}}</td>
63                         <td>{{records.desc}}</td>
64                         <td>{{records.op}}</td>
65                         <td>{{records.amount}}</td>
66                         <td>{{records.balance}}</td>
67                     </tr>
68                 </th>
69             </tr>
70         </tbody>
71     </table>
72 </div>
73 {% endblock content %}

```

2. Back-End Implementation

The back-end of the application is powered by **Flask**, a Python web framework. Flask handles the routing of HTTP requests and manages communication between the front-end and the database. It receives data from the user via form submissions, processes it, and stores it in the database. Additionally, Flask is responsible for generating financial reports and summaries.

A screenshot of a code editor window titled 'app.py'. The editor shows Python code for initializing a Flask application and setting up a database. The code includes imports for Flask, SQLAlchemy, datetime, and inspect. It then initializes the Flask app, sets the database URI to 'sqlite:///Finance.db', disables modification tracking, and creates a SQLAlchemy database instance.

```
1 from flask import Flask, redirect, render_template, request, url_for
2 from flask_sqlalchemy import SQLAlchemy
3 from datetime import datetime
4 from sqlalchemy import inspect
5
6 # Initialize the Flask application
7 app = Flask(__name__)
8
9 # Set up the database URI and configurations
10 app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///Finance.db' # Using SQLite database
11 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False # Disable modification tracking for performance
12 db = SQLAlchemy(app) # Initialize SQLAlchemy with the Flask app
```

```
56
57 # Route to handle the login page
58 @app.route('/', methods=['GET', 'POST'])
59 def home():
60     if request.method == 'POST':
61         AccountName = request.form.get('AccountName')
62         Password = request.form.get('Password')
63
64         user = User.query.filter_by(username=AccountName).first()
65         if user and user.password == Password:
66             return redirect(f'/{AccountName}')
67         else:
68             return redirect(f'/Register')
69
70     return render_template("home.html")
71
72
73 # Route to handle Registration
74 @app.route("/Register", methods=['GET', 'POST'])
75 def register():
76     if request.method == 'POST':
77         AccountName = request.form.get('AccountName')
78
79         # Register User in the database
80         new_user = User(username=AccountName, password=request.form.get('Password'))
81         db.session.add(new_user)
82         db.session.commit()
83
84         # Dynamically create a transactions table for the user
85         create_account(AccountName)
86
87         return redirect(f'/{AccountName}')
88
89     return render_template("register.html")
90
```

```

91
92 # Route to display the user's account page (authenticated users only)
93 @app.route('/<string:AccountName>', methods=['GET', 'POST'])
94 def show_account(AccountName):
95     model = create_account(AccountName)
96
97     if request.method == 'POST':
98         desc = request.form.get('Description')
99         withdraw_amount = request.form.get('Withdraw')
100         deposit_amount = request.form.get('Deposit')
101
102         last_transaction = model.query.order_by(model.sno.desc()).first()
103         current_balance = last_transaction.balance if last_transaction else 0
104
105         if withdraw_amount:
106             withdraw_amount = int(withdraw_amount)
107             if current_balance >= withdraw_amount:
108                 new_balance = current_balance - withdraw_amount
109                 new_withdraw = model(
110                     opr="Withdraw",
111                     desc=desc,
112                     amount=withdraw_amount,
113                     balance=new_balance
114                 )
115                 db.session.add(new_withdraw)
116                 db.session.commit()
117
118         if deposit_amount:
119             deposit_amount = int(deposit_amount)
120             new_balance = current_balance + deposit_amount
121             new_deposit = model(
122                 opr="Deposit",
123                 desc=desc,
124                 amount=deposit_amount,
125                 balance=new_balance
126             )
127             db.session.add(new_deposit)
128             db.session.commit()
129
130     transactions = model.query.all()
131     return render_template("account.html", AccountName=AccountName, transactions=transactions)
132

```

```

134 # Route for the 'About' page
135 @app.route("/About")
136 def about():
137     return render_template("about.html")
138
139
140 # Ensure that the database tables are created if they don't exist
141 if __name__ == "__main__":
142     with app.app_context():
143         db.create_all() # Create all tables (including User and any dynamic Account tables)
144
145     app.run(debug=True)
146

```


3. Data Storage and Management

A **relational database** (SQLite or SQL Server) is used to store user data securely. The database includes tables for storing transaction information, categories, and user details.

- **Transaction Table:** Stores details like transaction amount, date, category, and description.
- **Category Table:** Defines predefined categories (e.g., Food, Rent, Entertainment) to classify transactions.
- **User Table:** (Optional) Stores user credentials and preferences for personalized access.

SQL queries are used to interact with the database, including fetching transaction data, calculating totals, and generating reports.

```
15 # Dynamically create the Account model for each user
16 def create_account(table_name):
17     # Check if the table already exists in the database
18     inspector = inspect(db.engine)
19     if table_name in inspector.get_table_names():
20         # If the table exists, don't redefine the model.
21         class Account(db.Model):
22             __tablename__ = table_name
23             __table_args__ = {'extend_existing': True} # Ensure existing table is extended
24             sno = db.Column(db.Integer, primary_key=True)
25             date = db.Column(db.DateTime, default=datetime.utcnow)
26             desc = db.Column(db.String(500), nullable=True)
27             opr = db.Column(db.String(10), nullable=False)
28             amount = db.Column(db.Integer, nullable=False)
29             balance = db.Column(db.Integer, nullable=False)
30         return Account
31
32     # If the table doesn't exist, define the model dynamically and create it.
33     class Account(db.Model):
34         __tablename__ = table_name
35         __table_args__ = {'extend_existing': True} # Ensure existing table is extended
36         sno = db.Column(db.Integer, primary_key=True)
37         date = db.Column(db.DateTime, default=datetime.utcnow)
38         desc = db.Column(db.String(500), nullable=True)
39         opr = db.Column(db.String(10), nullable=False)
40         amount = db.Column(db.Integer, nullable=False)
41         balance = db.Column(db.Integer, nullable=False)
42
43     # Create the table if it doesn't already exist
44     db.create_all()
45     return Account
46
47
48 # Define the 'User' model for storing user login credentials
49 class User(db.Model):
50     __tablename__ = "Users"
51
52     id = db.Column(db.Integer, primary_key=True) # Primary key for user
53     username = db.Column(db.String(100), unique=True, nullable=False) # Unique username
54     password = db.Column(db.String(100), nullable=False) # Password for authentication
55
```

4. User Experience

The application is designed to be intuitive and easy to navigate. The user can:

- Add, edit, or delete transactions with just a few clicks.
- View and analyze financial data through interactive charts and graphs.
- Filter and search through transaction history to find specific entries.

By combining these technologies and techniques, the **Finance Tracker** is built to provide an efficient, secure, and user-friendly solution for managing personal finances.

2.5. Final Result

Finance TrackerAboutRegister

Account Name

Harsh

Please register first if you don't have an account.

Password

....

Login

Finance TrackerAboutRegister

Welcome! Harsh

Withdraw Amount

\$

Enter the amount...

.00

Description

Withdraw

Desposit Amount

\$

Enter the amount...

.00

Description

Deposit

S.No.	Date	Description	Action	Amount	Balance
1	2024-12-23 04:23:13.615026	Salary	Deposit	10000	10000
2	2024-12-23 04:25:25.134323	Rent	Withdraw	5000	5000
3	2024-12-23 04:25:48.351306	Grocery	Withdraw	2000	3000
4	2024-12-23 04:26:24.281963	Win Lottery	Deposit	500000	503000
5	2024-12-23 04:26:45.561045	Gaming Setup	Withdraw	80000	423000

3. Future Works

- **Advanced Budgeting and Forecasting:** Implement budgeting tools and forecasting features to track goals and predict future financial trends.
- **Bank Account Integration:** Automate transaction imports by integrating with bank accounts and payment services like PayPal.
- **Multi-Currency Support:** Enable tracking in multiple currencies with real-time exchange rate conversions.
- **Mobile Application:** Develop native iOS and Android apps for on-the-go finance management with push notifications.
- **Enhanced Analytics and Insights:** Integrate machine learning to provide personalized financial insights and recommendations.

4. Conclusion

The **Finance Tracker** is a powerful tool designed to help users manage their personal finances effectively. By allowing users to track income, expenses, and generate insightful reports, the application empowers individuals to make informed financial decisions. The use of a simple yet intuitive interface, coupled with data visualization tools, provides users with a clear understanding of their financial situation.

Through a combination of Flask, Python libraries like Pandas and Matplotlib, and a responsive front-end design, the application ensures a smooth and user-friendly experience. The ability to input, categorize, and analyze financial data in real-time makes the Finance Tracker an essential tool for personal finance management.

While the current version of the application meets fundamental needs, future enhancements like budgeting tools, bank integration, and advanced analytics will further enhance its value. With the potential for mobile apps, multi-currency support, and increased security, the Finance Tracker is poised to become an even more comprehensive solution for managing finances efficiently and securely.